

# Collaborative Continuous Benchmarking for HPC

MulticoreWorld'25  
February 18, 2025

Olga Pearce  
Lawrence Livermore National Laboratory  
Texas A&M University



LLNL-PRES-2202755

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

 Lawrence Livermore  
National Laboratory

# We benchmark HPC systems for a variety of reasons

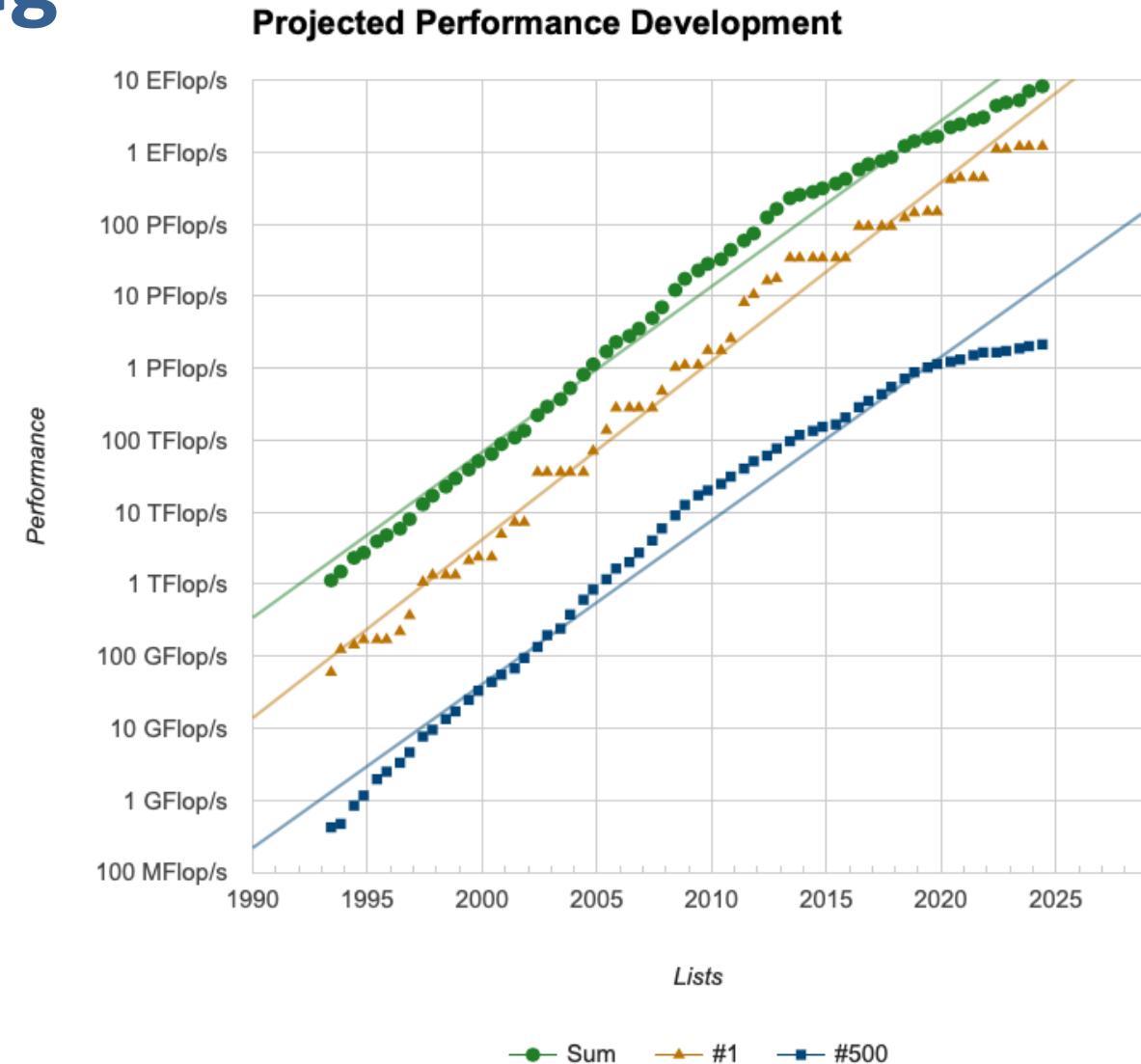
- Procurements
  - *Communicate* datacenter workload to vendors
  - Co-design systems, monitor progress
  - System acceptance (contractual spec)
- Validation of software stack, tools
  - Compilers
  - Debuggers
  - Correctness tools
  - Performance tools
- Research
  - Programming models
  - Computational methods
  - Performance across architectures



Many stakeholders, communication is key for collaboration

# Benchmarking is challenging

- What are we trying to characterize?
- Are we capturing the best the system can do?
- Is something else impacting performance?
- Did we build and run the code in the *optimal* and *reproducible* way?



# HPC benchmarks run on diverse HPC systems



Lawrence Berkeley Nat'l Lab  
AMD **Zen** + NVIDIA



Lawrence Livermore Nat'l Lab  
IBM **Power9** + NVIDIA



Oak Ridge National Lab  
AMD **Zen** + Radeon



Lawrence Livermore Nat'l Lab  
AMD **Zen** + Radeon



Argonne National Lab  
Intel **Xeon** + Xe

- Benchmark source code
  - Abstraction (OpenMP, RAJA, Kokkos)
  - Hardware-specific (CUDA, ROCm)
- Optimized code for the CPU and GPU
  - Must make effective use of the hardware
  - Can make 10-100x performance difference
- Rely heavily on system packages
  - Need to use optimized communication and MPI libraries that come with machines

# Writing benchmark source code is only the beginning

**State of the practice:**  
***HPC system benchmarking is manual!***

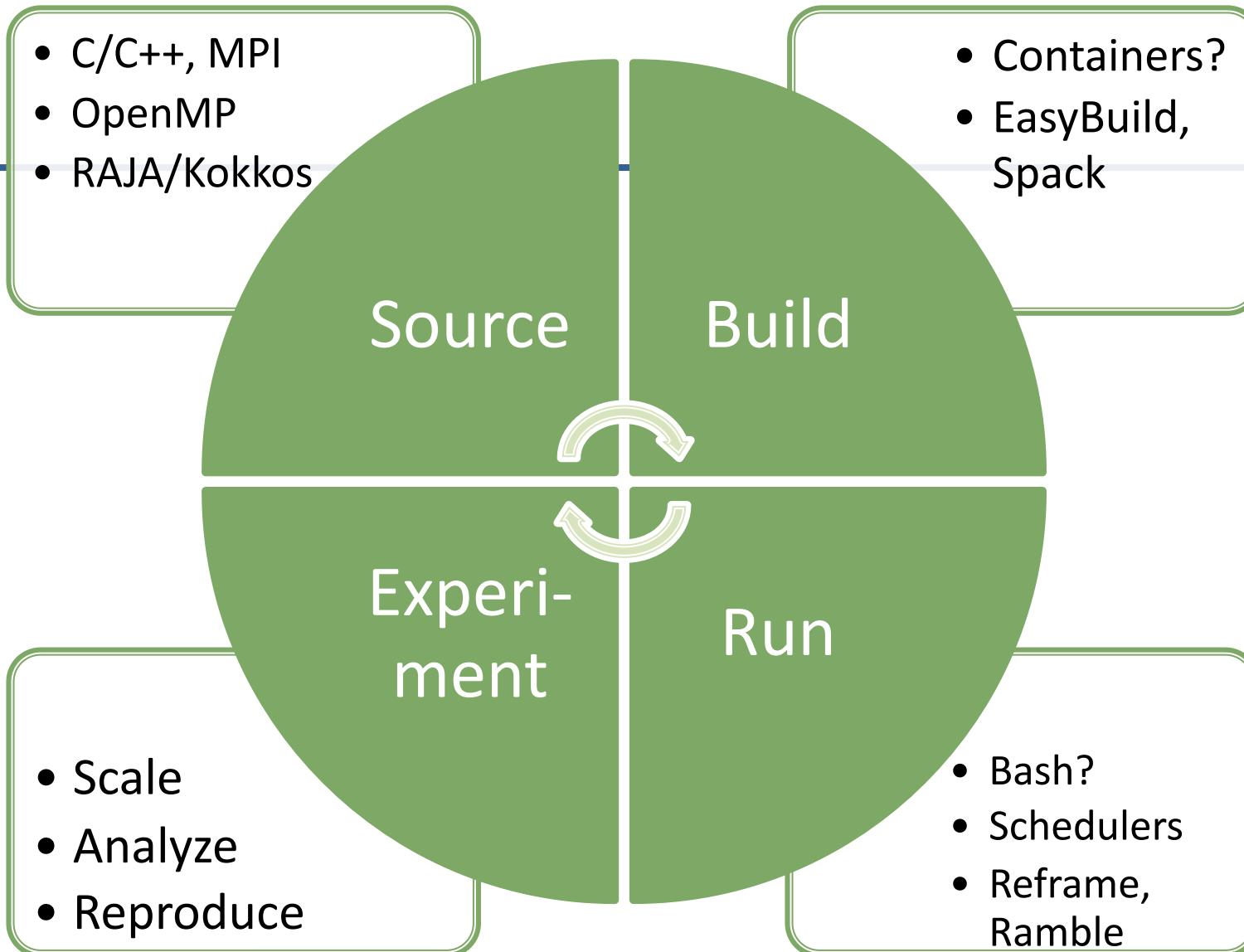


- **Building** on each system is different, porting the builds to new systems is manual
- **Running** on each system is different, porting run scripts to new systems is manual
- Systems keep **changing**, requiring updates to how we build and run benchmarks
- **Triggering** builds and runs is manual: benchmark results don't stay up to date
- **Performance analysis** of results is manual

**Communicate technical specification via code, documentation, white papers**

# HPC Benchmarks are HPC Software

- Portability
- Maintenance
- Testing/CI
- Verification
- Reproducibility



All components must work *for your system*, focus on explainable performance

# Benchpark enables complete specification of HPC benchmarks



*Infrastructure-as-code* benchmark specification codifies:

- Benchmark build and run instructions
- **HPC Systems**
- **HPC Experiments**

[github.com/llnl/benchpark](https://github.com/llnl/benchpark)

Leverage advances in HPC automation

- Source code
- Build specification
- Run specification
- CI



Every part of the specification codified: use to communicate, automate

# Benchpark enables *reproducible* specifications of benchmarks

## Specify

- How to build and run benchmarks on a system

## Run

- Run an experiment on a system

## Reproduce

- Re-run an experiment on a system

## Replicate

- Run an experiment on a new system

## Maintain

- CI: Run experiment on HPC systems

## Record

- Perf.measurements + *full spec* of experiment

Full specification enables reproducibility, replicability, and automation

# HPC System definition for *Performance*

- Resources (CPU cores, GPUs)
- Scheduler (Slurm, *flux*)
- Process mapping (cores, sockets, GPUs, NICs)
- On-node parallelism (CUDA, ROCM, OpenMP)
- Software stack
  - Compilers (which is best?)
  - MPI implementations (best?)
  - Math libraries



Goal: Use the system correctly

# HPC System in Benchpark: *Specify Once*

- ✓  systems
  - >  all\_hardware\_descriptions
  - >  aws-pcluster
  - >  generic-x86
  - >  lanl-venado
  - >  llnl-cluster
  - >  llnl-elcapitan
  - >  llnl-sierra
  - >  riken-fugaku

```
class LlnlElcapitan(System):  
    variant(  
        "rocm",  
        default="5.5.1",  
        values=("5.4.3", "5.5.1", "6.2.4"),  
        description="ROCM version",  
    )  
  
    variant(  
        "compiler",  
        default="cce",  
        values=("gcc", "cce", "rocmcc"),  
        description="Which compiler to use",  
    )  
  
    def initialize(self):  
        super().initialize()  
        self.scheduler = "flux"  
        self.sys_cores_per_node = "128"
```

`./benchpark system init --dest=elcap llnl-elcapitan rocm=6.2.4 compiler=cce`

**GETTING STARTED**

1. Getting Started with Benchpark
2. Searching Benchpark
3. Editing the experiment (optional)
4. Setting up Benchpark
5. Building the experiment
6. Running an Experiment in Benchpark
7. Analyzing Experiments in Benchpark

**FAQ**

What to rerun after edits  
Spack/Ramble versions in Benchpark  
Benchmark not yet in Spack/Ramble

**CATALOGUE****System Specifications****Benchmarks and Experiments****Benchpark Help Menu****CONTRIBUTING**

# System Specifications

The table below provides a directory of information for systems that have been specified in Benchpark. The column headers in the table below are available for use as the `system` parameter in [Benchpark setup](#). See [Setting up Benchpark](#) for more details.

	AWS_PCluster_Hpc7a-zen4-EFA	Fugaku	LUMI
site		RIKEN Center for Computational Science	CSC
system	AWS_PCluster_Hpc7a-zen4-EFA	Fujitsu-A64FX-TofuD	HPECray-zen3-MI250X-Slingshot
integrator.vendor	AWS	Fujitsu	HPECray
integrator.name	ParallelCluster3.7.2-Hpc7a	FX1000	EX235a
processor.vendor	AMD	Fujitsu	AMD
processor.name	EPYC-Zen4	A64FX	EPYC-Zen3
processor.ISA	x86_64	Armv8.2-A-SVE	x86_64
processor.uArch	zen4	aarch64	zen3
accelerator.vendor			AMD
accelerator.name			MI250X
accelerator.ISA			GCN
accelerator.uArch			gfx90a
interconnect.vendor	AWS	Fujitsu	HPECray

# Systems in Benchpark: February 2025

- 4 in Europe
- 5 in US labs
- 1 in Japan
- 1 at a university
- 2 Cloud systems

- CSC-LUMI-HPECray-zen3-MI250X-Slingshot
- CSCS-Daint-HPECray-haswell-P100-Infiniband
- CSCS-Eiger-HPECray-zen2-Slingshot
- JSC-JUWELS-Booster-rome-A100-Infiniband
- LLNL-Dane-DELL-sapphirerapids-OmniPath
- LLNL-Magma-Penguin-icelake-OmniPath
- LLNL-Pascal-Penguin-broadwell-P100-OmniPath
- LLNL-Ruby-icelake-OmniPath
- LLNL-Sierra-IBM-power9-V100-Infiniband
- LLNL-Tioga-HPECray-zen3-MI250X-Slingshot
- RCCS-Fugaku-Fujitsu-A64FX-TofuD
- TAMU-Grace-Dell-cascadelake-Infiniband
- common
- nosite-AWS\_PCluster\_Hpc6a-zen3-EFA
- nosite-AWS\_PCluster\_Hpc7a-zen4-EFA
- nosite-HPECray-zen3-MI250X-Slingshot

# HPC Experiment definition for *Performance*

- On-node parallelism  
(CUDA, ROCM, OpenMP)
- Problem sizes
  - Overall problem size, or
  - Per node or per GPU
- Scaling studies
  - How to scale
  - How to decompose
- Resources (cores, GPUs)



**Goal: Specify reproducible sets of experiments that map onto specific Systems**

# HPC Experiment in Benchpark: *Specify Once*

```
class Amg2023(Experiment, OpenMPExperiment, CudaExperiment, ROCmExperiment,
               StrongScaling, WeakScaling, ThroughputScaling, Caliper):
    variant(
        "workload",
        default="problem1",
        values=("problem1", "problem2"),
        description="problem1 or problem2",
    )

    def compute_applications_section(self):
        scaling_modes = {
            "strong": self.spec.satisfies("+strong"),
            "weak": self.spec.satisfies("+weak"),
            "throughput": self.spec.satisfies("+throughput"),
            "single_node": self.spec.satisfies("+single_node"),
        }
```

amg2023 +rocm +strong workload=problem2 caliper=mpi,time

**GETTING STARTED**

1. Getting Started with Benchpark
2. Searching Benchpark
3. Editing the experiment (optional)
4. Setting up Benchpark
5. Building the experiment
6. Running an Experiment in Benchpark
7. Analyzing Experiments in Benchpark

**FAQ**

What to rerun after edits  
Spack/Ramble versions in Benchpark  
Benchmark not yet in Spack/Ramble

**CATALOGUE**

System Specifications

**Benchmarks and Experiments**

Benchpark Help Menu

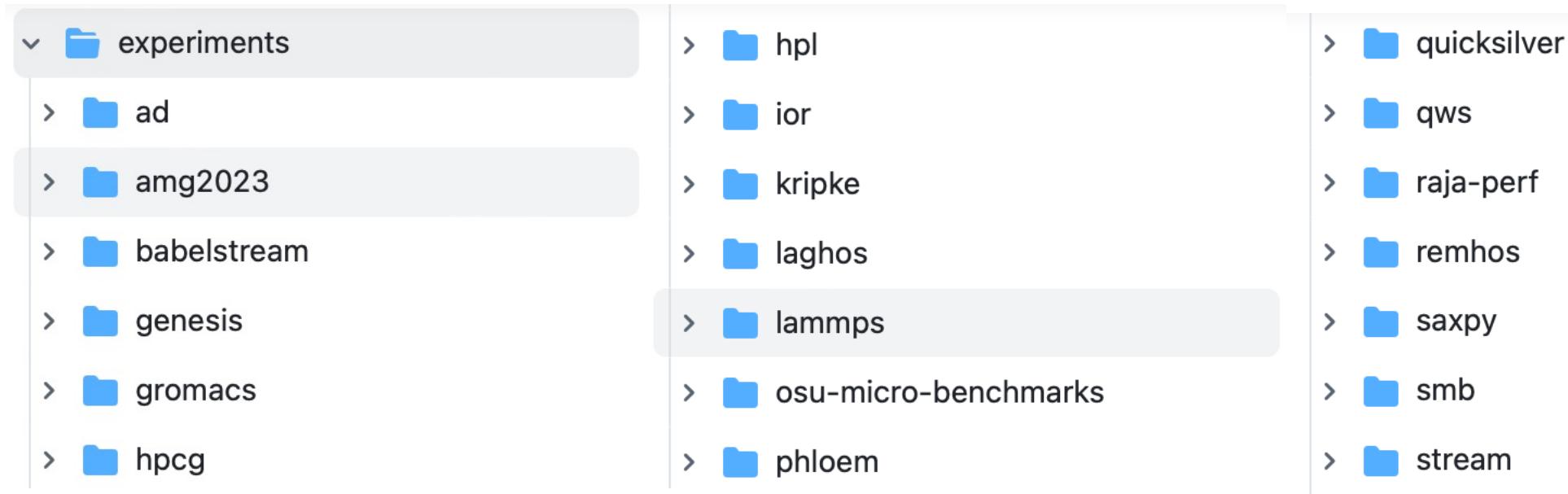
**CONTRIBUTING**

# Benchmarks and Experiments

	<b>hpl</b>	<b>hpcg</b>	<b>qws</b>
application-domain	['synthetic']	['synthetic']	['qcc']
benchmark-scale	['large-scale']	['large-scale']	['we']
communication	['mpi']	['mpi']	['mp']
communication-performance-characteristics	['network-collectives', 'network-point-to-point']	['network-point-to-point']	[]
compute-performance-characteristics	[]	[]	[]
math-libraries	['blas']	[]	[]
memory-access-characteristics	[]	[]	[]
mesh-representation	[]	[]	[]
method-type	['dense-linear-algebra', 'solver']	['conjugate-gradient', 'solver', 'sparse-linear-algebra']	[]
programming-language	['c']	['c++']	['c++']
programming-model	[]	['openmp']	[]

[◀ Previous](#)[Next ▶](#)

# Experiments in Benchpark: February 2025



- HPL, HPCG
- 4 microbenchmarks
- 3 MPI benchmarks
- 8 US
- 1 Europe
- 2 Japan

# Benchpark is 18 months old

## Contributions from 11 orgs (60% non-LLNL)

Product ▾ Solutions ▾ Resources ▾ Open Source ▾ Enterprise ▾ Pricing

Search or jump to... / Sign in

LLNL / benchpark Public

Notifications Fork 26 Star

Code Issues 69 Pull requests 38 Discussions Actions Projects Security Insights

Pulse

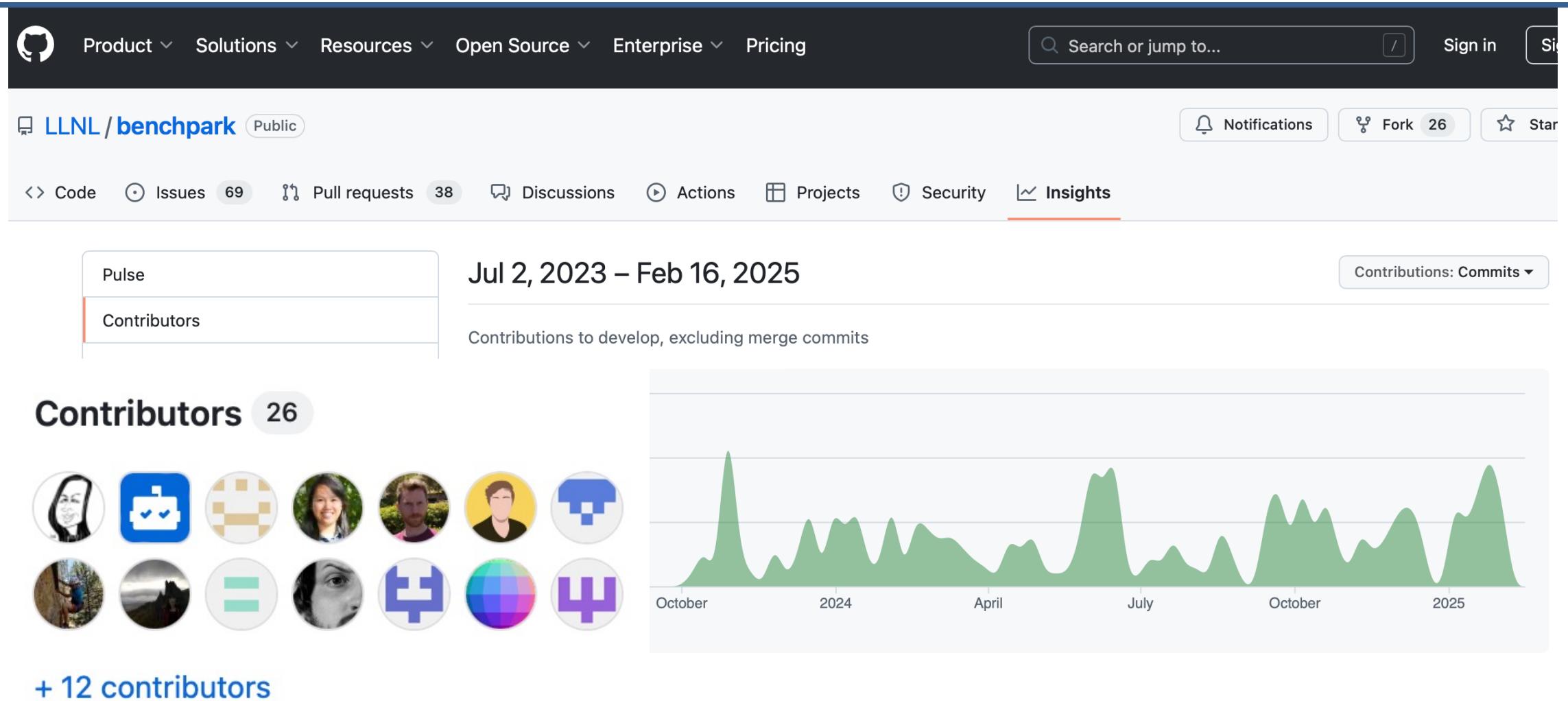
Contributors

Jul 2, 2023 – Feb 16, 2025

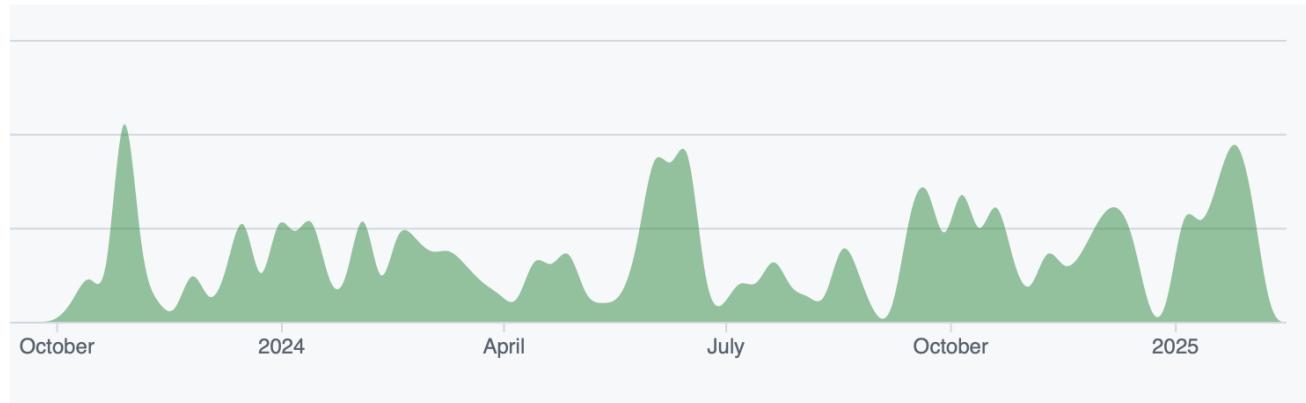
Contributions: Commits

Contributors 26

+ 12 contributors



Contributions to develop, excluding merge commits



# Benchpark codifies benchmarking steps

- Benchpark does **not** replace benchmark source, build system, or Spack package
- Benchpark manages benchmark **experiments** and how they map onto **systems** with specified (or default)
  - Compilers
  - MPI/comm libraries
  - Math libraries
- Start running benchmark experiments on your system with just a few commands

✉ git clone git@github.com:LLNL/benchpark.git

🖥 benchpark list systems  
benchpark system init --dest=elcap llnl-elcapitan

benchpark list benchmarks  
👤 benchpark experiment init dest=amg  
amg2023+rocm+strong

💻 benchpark setup amg elcap workspace  
ramble workspace setup

🐦 ramble on

# Who can use Benchpark

**People who want to use or distribute benchmarks for HPC!**

## **1. End Users of HPC Benchmarks**

- Install, run, analyze performance of HPC benchmarks

## **2. Benchmark Developers**

- People who want to share their benchmarks

## **3. Procurement teams at HPC Centers**

- Curate workload representation, evaluate and monitor system progress

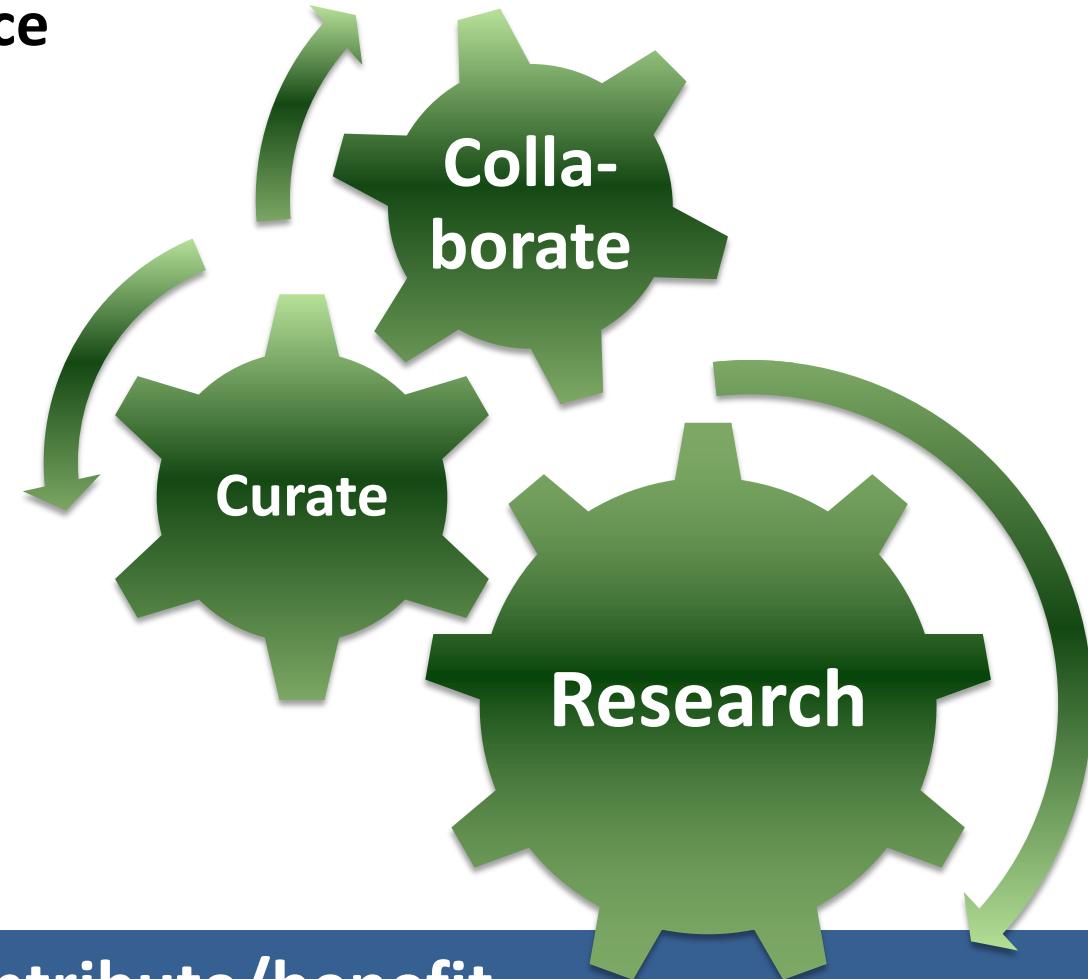
## **4. HPC Vendors**

- Understand the curated workload of HPC centers, propose systems

# Catalogued library of *working* benchmarks



- Enables exploration of large configuration space
  - Architectural
  - Software stack
  - Temporal
- What architecture and system configuration is best for *my benchmark*?
- On *my system*, is OpenMPI good enough?
- What purpose will Benchpark help you address?



Building a community to contribute/benefit

# Benchpark: Open collaborative repository for *reproducible* specifications of HPC benchmarks



*Infrastructure-as-code* benchmark specification enables **reproducibility, replicability, and automation**

- **HPC Systems**
- **HPC Experiments**

## Specify

- How to build and run benchmarks on a system

## Run

- Run an experiment on a system

## Reproduce

- Re-run an experiment on a system

## Replicate

- Run an experiment on a new system

## Maintain

- CI: Run experiment on HPC systems

## Record

- Perf.measurements + *full spec* of experiment

- Tagging, keywords for publications
- Performance metrics, metrics of usefulness
- Dashboards: Archive specs+results
- CI pipelines on PRs from GitHub at data centers across the world and in the cloud

\*Olga Pearce et al, Continuous Benchmarking, HPCTests SC|23

**Full specification enables reproducibility, replicability, and automation**

# Benchpark roadmap and community engagement

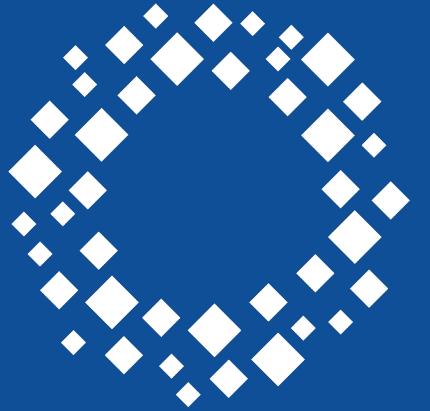
## Future directions:

- Suite curation: Reproducible specification of an entire suite
- Tagging: Keywords for publications, finding benchmarks
- Metrics: Performance, usefulness
- Dashboards: Archive+share *specs+results*, Slices in configuration space
- CI pipelines at data centers across the world and in the cloud

## Community Engagement:

- Co-design / vendors on board, but incentive for app teams? (carrot or stick?)
- Who owns which parts of the specification and approves changes?
- Who finances R&D and maintenance?
- ROI for the people working on it? → think about post docs, researchers, etc.

**Collaboration, reproducibility, fully specified public results**



# CASC

Center for Applied  
Scientific Computing



**Lawrence Livermore  
National Laboratory**

#### **Disclaimer**

This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.